Time-Shift Scheduling—Fair Scheduling of Flows in High-Speed Networks

Jorge A. Cobb, Mohamed G. Gouda, and Amal El-Nahas

***Abstract*—We present a scheduling protocol, called time-shift scheduling, to forward packets from multiple input flows to a single output channel. Each input flow is guaranteed a predetermined packet rate and an upper bound on packet delay. The protocol is an improvement over existing protocols because it satisfies the properties of rate-proportional delay, fairness, and efficiency, while existing protocols fail to satisfy at least one of these properties. In time-shift scheduling each flow is assigned an increasing timestamp, and the packet chosen for transmission is taken from the flow with the least timestamp. The protocol features the novel technique of time shifting, in which the scheduler's real-time clock is adjusted to prevent flow timestamps from increasing faster than the real-time clock. This bounds the difference between any pair of flow timestamps, thus ensuring the fair scheduling of flows.**

***Index Terms*—Quality of service, real-time network protocols, real-time scheduling.**

I. INTRODUCTION

CONSIDER a computer network with point-to-point communication channels. Assume that a source computer wishes to transfer a sequence of packets to a destination computer. We call such a sequence a flow; that is, a flow is a sequence of packets generated by the same source and addressed to the same destination. Assume also that the source computer requires a lower bound on the rate at which its packets are forwarded through the network, and an upper bound on the packet delay from source to destination.

To solve this problem, a particular type of scheduling protocols, which we call rate-reservation protocols, were developed to forward packets from each flow at a designated rate. Examples of these protocols can be found in [26] and [27]. In these protocols the source of a flow finds a network path that leads to its desired destination. Then, it notifies each computer in the path about its desired packet rate. Each computer determines if it has enough available bandwidth in its output channel to forward the packets from the new flow. The new flow is accepted if and only if all computers in the path accept the new flow.

Manuscript received March 24, 1997; revised October 8, 1997 and December 5, 1997; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Lee.

J. A. Cobb is with the Department of Computer Science, University of Houston, Houston, TX 77204-3475 USA (e-mail: cobb@cs.uh.edu).

M. G. Gouda is with the Department of Computer Sciences, University of Texas, Austin, TX 78712-1188 USA (e-mail: gouda@cs.utexas.edu).

A. El-Nahas is with the Department of Computer Science, Alexandria University, Alexandria, Egypt (e-mail: amal@dataxprs.com.eg).

Publisher Item Identifier S 1063-6692(98)04080-1.

Due to the reservation of bandwidth, the network can provide service guarantees to each flow, such as end-to-end packet delays, provided the rate of the flow does not exceed the reserved rate. These service guarantees are of particular importance to real-time applications, such as interactive audio and video [11].

The desirable properties of rate-reservation protocols are the following.

1) *Rate-proportional delay:* Let f be an input flow of a computer, with a reserved rate of $R.f$ bits/s. Assume that f is also the sole input to a constant rate server that forwards the bits of each packet of f at a precise rate of $R.f$ bits/s. The delay of each packet of f through the computer should be at most the delay of the same packet through the constant rate server (plus a small constant).
2) *Efficiency:* The time to enqueue a received packet or to dequeue a packet for transmission is $O[\log(N)]$, where N is the number of flows sharing the output channel.
3) *Fairness:* A flow should not be "punished" if it temporarily exceeds its reserved rate to take advantage of unused bandwidth in the channel. In addition, unused bandwidth should be shared among the flows in proportion to their reserved rates.

The rate-proportional delay property guarantees to each flow that the upper bound on its packet delay depends solely on its reserved rate and not on other factors, such as the number of flows sharing the output channel or the reserved rate of other flows. The efficiency property is desirable due to the high bandwidth requirements expected from future applications of rate-reservation protocols.

The fairness property is desirable because it may be normal for some flows to violate their reserved packet rate. Examples of such flows are file transfers and multiresolution video [19]. The sources of these flows may reserve from the network the smallest packet rate necessary to receive a minimum quality of service. If the source of a flow detects that additional bandwidth is available, then it generates packets at a rate higher than its reserved rate in order to take advantage of the unused bandwidth. If the source detects that no additional bandwidth is available, it reduces its sending rate. (There are several techniques by which a source can detect if additional bandwidth is available, see, for example, [17] and [23].) Thus, because some flows may be of adjustable rate, the unused bandwidth of a channel should be shared in a fair manner among all flows traversing the channel.

Some rate-reservation protocols are inadequate for adjustable rate flows because they are not work conserving

[10], [13], [18], [22]. That is, they serve each flow at exactly the rate it reserved, and will not forward additional packets of the flow even if the outgoing channel is idle. These protocols do not allow adjustable rate flows to take advantage of any additional bandwidth and, thus, do not satisfy the fairness property.

Other rate-reservation protocols assign a timestamp to each packet, and packets are forwarded in increasing timestamp order. These protocols are work conserving; that is, the output channel is never idle as long as its packet queue is nonempty. However, some of these protocols are unfair [25] in the sense that they "punish" a flow if it sends packets at a rate higher than its reserved rate. Other protocols are fair [14], [21], but they are either inefficient or do not have rate-proportional delay.

In this paper we introduce a new rate-reservation protocol called time-shift scheduling [4]. The protocol is based on flow timestamps (i.e., assigning a timestamp per flow rather than per packet), is work conserving, and satisfies all of the desirable properties mentioned above. Time-shift scheduling is based on the novel technique of time shifting, in which the real-time clock is periodically adjusted to prevent flow timestamps from increasing faster than the real-time clock. This ensures fairness by placing an upper bound on the difference between the timestamps of any pair of flows.

The paper is organized as follows. In Section II we discuss the use of flow timestamps in rate-reservation protocols. In Section III we present the time-shift scheduler, and its formal definition is given in Section IV. We show that the time-shift scheduler has rate-proportional delay in Section V, and its fairness is shown in Section VI. In Section VII we derive the end-to-end delay bounds for a path of time-shift schedulers. Future work is given in Section VIII.

Notation: Throughout the paper we use quantifications of the form

$$[\otimes x : R(x) : B(x)].$$

Above, $\otimes$ is a commutative and associative operator, such as $\max$, $\min$, $\sum$ (summation), $\forall$ (conjunction), or $\exists$ (disjunction). $R(x)$ is a Boolean function defining the range of values for the dummy variable x, and $B(x)$ is a function defining the value given as an operand to $\otimes$. For example

$$(\min\ x : 1 \le x \le 3 : x^2)$$

denotes the minimum of 1^2, 2^2, and 3^2. If $R(x)$ is omitted, all values in the type of x are included.

II. Flow Timestamps

In this section we discuss the strengths and weaknesses of the scheduling protocols of virtual clock [25], weighted fair queueing [17], [20], [21], and self-clocking fair queueing [14]. Before doing so, we present some background on the scheduling technique used by these protocols.

A *computer network* consists of a set of computers interconnected via point-to-point bidirectional channels. A *flow* in a computer network is a potentially infinite sequence of packets generated by the same source and having the same destination

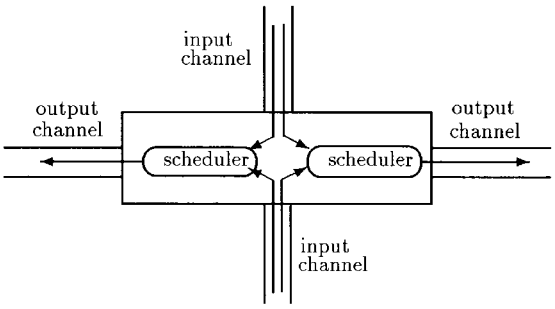


Fig. 1. A computer with two input channels and two output channels.

in the network. When a new flow wishes to join the network, the network finds a path from the source of the flow to the destination of the flow. Then, the network reserves a fraction of the packet rate of each channel along the path and assigns this rate to the new flow. Finally, the source of the flow is given permission to generate packets at the rate reserved for it by the network. The chosen path and reserved rate of the flow remain fixed throughout the lifetime of the flow.

Each output channel of a computer is equipped with a *scheduler*, as shown in Fig. 1. From the input channels, the scheduler receives packets from flows whose next hop to the destination is the output channel of the scheduler. Whenever its output channel becomes idle, the scheduler chooses a received packet and forwards the packet to the output channel. The rate at which the scheduler forwards the packets of a flow must be bounded from below by the reserved rate of the flow. To guarantee this minimum packet rate, the scheduler assigns a timestamp to each received packet. The timestamp is a function (among other things) of the flow's reserved rate. The scheduler then forwards the packets in order of increasing timestamp.

The scheduler maintains a separate FIFO queue for the received packets from each flow. We say that a flow is *active* if its queue in the scheduler is nonempty.

We have shown in [3] that assigning a timestamp to a packet when it becomes the head of the queue of its flow is cleaner and more efficient than the original method of assigning a timestamp to the packet when it is received. We refer to the former technique as *flow timestamps*, since only one timestamp is maintained per flow. Thus, we base time-shift scheduling on flow timestamps. To maintain a single timestamp paradigm throughout the paper, the flow timestamp versions of the virtual clock, self-clocking fair queueing, and weighted fair queueing protocols will be discussed.

We adopt the following notation for a scheduler:

N	number of flows in the scheduler;
$\mathrm{queue}.f$	queue of received packets from flow f;
$R.f$	forwarding rate (in bits per second) reserved for flow f;
$L.f$	packet length (in bits) of the head of $\mathrm{queue}.f$;
$T.f$	timestamp of flow f;
$L.p$	packet length (in bits) of packet p;
$T.p$	value of $T.f$ when packet p is at the head of $\mathrm{queue}.f$, where f is the flow of packet p;
$L_{\max}$	upper bound on packet length for all flows;
C	capacity in bits per second of the output channel.

We say that a packet is *forwarded to the output channel* by the scheduler when the first bit of the packet is being transmitted by the output channel. We say that a packet *exits the output channel* when the last bit of the packet is transmitted by the output channel. We say that a packet is *in the output channel* if it has been forwarded to the output channel but has not yet exited the output channel.

The goal of the scheduler is to forward the packets of each flow f at an average rate of at least $R.f$. Since all N flows share the output channel, the following constraint is necessary:

$$\sum_{f=0}^{N-1} R.f \leq C. \tag{1}$$

Assume packet p from flow f is received. Before the packet is appended to queue.f, the scheduler checks if f is active. If f is not active, $T.f$ is updated as follow:

$$T.f := \max(v.p, T.f) + L.p/R.f.$$

In this assignment, $v.p$ is some quantity related to packet p. The value chosen for $v.p$ varies from one scheduling protocol to another. On the other hand, if f is active, $T.f$ is not updated.

When the output channel becomes idle, the scheduler finds the active flow with the smallest timestamp. Let f be this flow. Then, the next packet from f is removed from its queue and forwarded to the output channel. If f remains active, its flow timestamp is updated as follows:

$$T.f := T.f + L.f/R.f.$$

We next consider the case of virtual clock scheduling, which is defined by choosing $v.p$ as the time at which p is received by the scheduler. Virtual clock has the property of rate-proportional delay, as proven in [3] and [24]. In particular, the exit time of p is at most $T.p + L_{\max}/C$. The scheduler is also efficient, requiring only $\mathrm{O}[\log(N)]$ operations to enqueue or dequeue a packet. However, the scheduler is unfair, as illustrated by the following well-known example.

Let packet sizes be constant, and the output channel have a rate of 1 packet/s. The scheduler has two flows f and g, each with a reserved rate of 1/2 packet/s. Consider the following sequence of events.

From time 0 up to time 100 s, packets from flow f arrive at a rate higher than 1 packet/s, and no packet is received from flow g. Thus, at time 100 s, 100 packets from f have been forwarded to the output channel, the queue of f is not yet empty, $T.f = 202$ s, and $T.g = 0$.

At time 100 s, packets from g arrive at a rate of at least 1 packet/s, and packets from f continue to arrive. Note that when the next packet from flow g is received, $T.g = 102$ s. Since $T.f$, which equals 202 s, is much larger than $T.g$, no packet from f is forwarded to the output channel until 50 packets from g are forwarded, i.e., until time 150 s. In effect, f is denied service for 50 s because it earlier took advantage of bandwidth unused by g.

This unfairness does not occur in the weighted fair queueing protocol. Furthermore, the bound on packet delay is similar to that of virtual clock and, thus, it also satisfies the rate-proportional delay property. In weighted fair queueing, the timestamp of a packet is the time at which the packet would exit a virtual server. The input to the virtual server are the same input flows of the scheduler, and the virtual server shares its unused bandwidth among all active flows in proportion to their reserved rates. The value of $v.p$ is complicated and takes $\mathrm{O}(N)$ time to compute.

The higher complexity of weighted fair queueing led to the introduction of self-clocking fair queueing. In this scheduler, $v.p = T.q$, where q is the packet in the output channel at the time p is received. The time to enqueue or dequeue a packet is $\mathrm{O}[\log(N)]$, as it is in virtual clock.

This protocol is fair in the following sense. For any pair of active flows f and g

$$|T.f - T.g| \leq \max(L.f/R.f, L.g/R.g).$$

In this way no flow can have a timestamp that is significantly greater than the timestamp of other flows. Thus, a flow that takes advantage of free bandwidth cannot be punished. On the other hand, in virtual clock scheduling, $|T.f - T.g|$ is unbounded.

Unfortunately, the packet delay increases as follows. Consider the same scheduler as above, with a fixed packet size and an output channel with rate 1 packet/s. Let the scheduler have 91 flows. Flows 1–90 have a rate of 1/100 packets/s, and flow 0 has a rate of 1/10 packets/s. Assume that one packet from each of flows 1–90 arrive at time 0. The timestamp of each of these packets is 100 s. Next, when the first of these packets is in the output channel, a packet from flow 0 arrives. The timestamp of this packet is 110 s. This packet must wait until all 90 packets with timestamp 100 s exit the output channel before it may exit. Hence, the delay of this packet is 91 seconds, as opposed to a delay of at most 11 seconds that it would incur in virtual clock or weighted fair queueing.

Note that the delay of flow 0 is related to the rate of the other flows. If the 90 flows with a rate of 1/100 packet/s are replaced by 900 flows with a rate of 1/1000 packet/s, the delay of flow 0 increases by a factor of ten.

We conclude that each of the scheduling protocols discussed above satisfies only two out of the three desired properties. We next present a scheduling protocol that satisfies all three properties.

III. Time-Shift Scheduling

We next present the intuition behind time-shift scheduling. Its formal definition, its delay, and its fairness properties are given in later sections.

From its definition, $T.f$ can be viewed, intuitively, as the time at which the packet at the head of the queue of flow f should exit the output channel. That is, it should exit $L.f/R.f$ seconds later than the exit time of the previous packet from the same flow. Thus, we may define the "ideal arrival time" of the packet at the head of the queue of flow f as follows:

$$I.f = T.f - L.f/R.f.$$

For example, if the packet should exit at time $T.f$ and the flow is abiding by its reserved rate $R.f$, then, ideally, the packet should be received at time $I.f$.

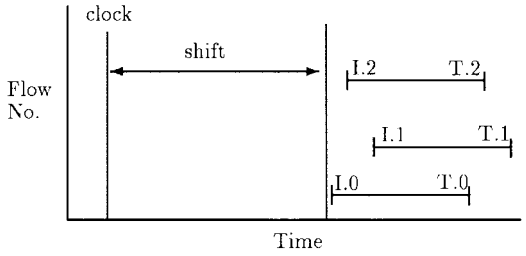


Fig. 2. Drift of flow timestamps from the real-time clock.

Consider Fig. 2, in which flows 0–2 are active and the vertical line on the left indicates the current value of the real-time clock. Flow timestamp $T.f$ is an indicator of how much service has been given to flow f. If $T.f \gg \text{clock}$, then packets from f have been forwarded at a rate greater than $R.f$. This could occur if the flow is trying to take advantage of unused bandwidth. In the figure, all three flows have a flow timestamp significantly greater than the current value of the clock.

Assume that an additional flow, flow 3, has been inactive for some time but becomes active once again. Assume we choose $v.p$ to be the arrival time of packet p. Then, when flow 3 becomes active, its flow timestamp is updated as follows:

$$T.3 := \max(\text{clock}, T.3) + L.3/R.3.$$

Thus, $T.3 = \text{clock} + L.3/R.3$, which is significantly smaller than the flow timestamps of the other active flows. This implies that only packets from flow 3 are forwarded until $T.3$ reaches a value greater than the flow timestamps of the other flows.

To remedy this, $T.3$ should be given a value as close as possible to $T.0$, $T.1$, and $T.2$. Since $T.3$ is derived from the real-time clock, the clock should be close to $T.0$, $T.1$, and $T.2$. To do this, either we increase the value of the real-time clock, or we reduce the value of each flow timestamp by an equal amount. We take the former approach, because the latter requires $\mathrm{O}(N)$ time.

The remaining question is how much to advance the clock. If the clock is advanced beyond the minimum of the ideal arrival times, then the flow that becomes active has a timestamp larger than the timestamps of the active flows and may be delayed excessively by these flows. If the clock is advanced to a value smaller than the minimum of the ideal arrival times, then the flow that becomes active has a timestamp smaller than the timestamps of the active flows, and it may unfairly delay the active flows. Thus, we choose to advance the clock to the minimum of the ideal arrival times of the active flows.

Because the scheduler advances the clock, i.e., it "shifts" the clock to the right, it is called a time-shift scheduler. We refer to the adjustable real-time clock by the name ShiftClock.[1]

Whenever a flow becomes active, the scheduler performs a time shift by updating ShiftClock as follows:

$$\text{ShiftClock} := \max(\text{ShiftClock}, I_{\min})$$

[1]If the clock of the scheduler cannot be advanced, the scheduler may increment a variable, say *shift*, instead of incrementing the clock. Any reference to the clock is then substituted by the expression *clock* + *shift*.

where $I_{\min}$ is the minimum ideal arrival time of all active flows, that is

$$I_{\min} = (\min\ f : \text{queue}.f \neq \textbf{empty} : I.f).$$

The restriction on the reserved rates given by (1) above is sufficient, provided the rate assigned to each flow remains fixed. However, this is not always the case, because the application generating packets for the flow may terminate and the flow may be reassigned to a new application that reserves a different rate. Thus, a restriction is needed to indicate when can the reserved rate of a flow be reassigned to another flow.

The restriction we choose is the following. We say that a flow f is *live* if either $\text{ShiftClock} \leq T.f$ or f is active. The rate of a flow f can be reassigned to another flow if flow f is no longer live. Thus, the following is required to be an invariant of the reserved rates:

$$\left(\sum f : \text{ShiftClock} \leq T.f \wedge \text{queue}.f \neq \textbf{empty} : R.f\right) \leq C. \tag{2}$$

Next, we revisit the scenarios for virtual clock and self-clocking fair queueing discussed in Section II, and see how unfairness and long delays are avoided in the case of time-shift scheduling. Recall that in both scenarios the packet size is constant, and the output channel has a capacity of 1 packet/s.

In the first scenario, we have two flows f and g, each with a reserved rate of 1/2 packet/s. From time 0 up to time 100 s, packets from flow f arrive at a rate higher than 1 packet/s, and no packet is received from g. Thus, at time 100 s, 100 packets from f have been forwarded to the output channel, the queue of f is not empty, $T.f = 202$ s, and $T.g = 0$.

At time 100 s, packets from g arrive at a rate of 1/2 packet/s (or any rate higher than this) and packets from f continue to arrive. When the first packet of g arrives at time 100 s, a time shift is performed because g becomes active. Thus, $\text{ShiftClock} = I_{\min} = I.f = 200$ s and $T.g$ is assigned 202 s, which equals $T.f$. Therefore, from this point onward, one packet of g is forwarded for every packet of f, and the unfairness experienced with virtual clock is avoided.

In the second scenario we have 100 flows. Flow 0 has a rate of 1/10 packets/s, and flows 1–99 have a rate of 1/100 packets/s. At time 0, a packet from each of flows 1–99 is received, and the timestamp of each of these is 100 s. After the first of these packets is forwarded to the output channel, a packet from flow 0 arrives. Since flow 0 is becoming active, a time shift is performed and $\text{ShiftClock} := \max(\text{ShiftClock}, I_{\min})$. Since $I_{\min} = 0$ and the first packet is still in the channel, $0 \leq \text{ShiftClock} \leq 1$. Hence, the timestamp of flow 0 is at most 11 s. Since the other flows have a timestamp of 100 s, the next packet to be forwarded is from flow 0. Thus, this packet experiences a delay of at most 2 s, rather than the delay of 91 s that it would experience in self-clocking fair queueing.

IV. Protocol Specification

We next provide a more formal description of a time-shift scheduler. We define the behavior of a scheduler process by a set of global constants, a set local inputs, a set of local

variables, and a set of actions. Actions are separated from each other with the symbol [], using the following syntax:

$$\textbf{begin}\ \text{action}\ [] \ \text{action}\ [] \cdots [] \ \text{action}\ \textbf{end}.$$

Each action is of the form *guard* $\rightarrow$ *command*. A guard is either a Boolean expression involving the local variables of its process or a receive statement of the form **receive** p **from any** f that receives a packet from any input flow. A command is constructed from sequencing (;) and conditional (**if fi**) constructs that group together **skip**, assignment, and statements of the form **forward** p, where p is a packet. Similar notations for defining network protocols are discussed in [15] and [16].

An action in a scheduler process is said to be *enabled* if its guard is either a Boolean expression that evaluates to **true** or a receive statement of the form **receive** p **from any** f, and there is a packet that may be received from some input flow.

An execution step of a protocol consists of choosing any enabled action from the process and executing the action's command. If the guard of the chosen action is a receive statement **receive** p **from any** f, then, before the action's command is executed, the packet is stored in variable p and its flow number is stored in variable f. If the statement to execute in the command is of the form forward p, then packet p is forwarded to the output channel.

Protocol execution is fair; that is, each action that remains continuously enabled is eventually executed.

The specification of the time-shift scheduler is presented in Fig. 3. The process has two inputs from its environment. The first is a Boolean bit which indicates if the output channel is currently idle. It becomes false when the scheduler forwards a packet p to the output channel, and becomes true $L.p/C$ s later. The second input is the rate reserved for each flow.

We assume the following. Variable ShiftClock is an adjustable real-time clock. It increases automatically with the progression of time. Also, executing an action takes zero time, i.e., ShiftClock remains constant while an action is executed unless an assignment statement in the action changes its value. Finally, ShiftClock does not advance while the packet queue is nonempty and the output channel is idle, i.e., the next packet to forward is chosen immediately after the output channel becomes idle.

The time-shift scheduler process may be specified as follows.

The process contains three actions. In the first action a packet is received from a flow. If the flow becomes active, then the flow timestamp is updated.

In the second action, when the output channel is idle and there are still packets to forward, the active flow with the smallest timestamp is obtained from function least (T) and a packet from this flow is forwarded. The flow's timestamp is updated if the flow remains active. If no active flows remain, ShiftClock is updated so that it is greater than the flow's timestamp. This is necessary to prove fairness in Section VI.

In the third action a time shift is performed, provided there is at least one active flow. Note that the scheduler has a lot of freedom in choosing when to perform a time shift, i.e., it could be done often or seldom. Regardless of when a time shift is performed, the property of rate-proportional delay is satisfied. However, if fairness is desired, the scheduler must execute a time shift whenever a flow becomes active. That is, immediately before executing the first action for an inactive flow f, the third action must be executed. We will show the correctness of these statements in the sections that follow.

```
process Time-Shift Scheduler
inputs
idle        : is the output channel idle?
R.f         : rate of flow f
variables
p           : packet
L.p         : length of packet p
f, g        : 0 . . N-1
queue.f     : packet queue of flow f
T.f         : timestamp of flow f
ShiftClock  : adjustable real-time clock
begin
 receive p from any f  →
      if queue.f = empty  →
                T.f := max(ShiftClock, T.f) + L.p/R.f
      [] queue.f ≠ empty  →
                skip
      fi;
      queue.f := append(queue.f, p)
[]
 idle ∧ ( ∃ g :: queue.g ≠ empty)  →
      f := least(T);
      p := head(queue.f);
      forward p;
      queue.f := tail(queue.f);
      if queue.f ≠ empty  →
                T.f := T.f + L.p/R.f
      [] (∀ g :: queue.g = empty)  →
                ShiftClock := max(ShiftClock, T.f)
      [] (∃ g :: queue.g ≠ empty) ∧
        queue.f = empty  →
                skip
      fi
[]
 (∃ g :: queue.g ≠ empty ) ∧ ShiftClock < I_min  →
      ShiftClock := I_min
end
```

Fig. 3. Specification of the time-shift scheduler.

Implementing this protocol requires two ordered queues: one for the flow timestamps and one for the ideal service times. Inserting or removing an element from either of these takes $O[\log(N)]$ time. Thus, the desired efficiency is achieved.

V. Local Delay Bound

In this section we show that the time-shift scheduler has a bound on packet delay no greater than the bound on packet delay of a virtual clock scheduler or a weighted fair queueing scheduler.

Henceforth, any reference to time refers to the value of ShiftClock and not to the true value of real time. For example, the expression "at time t" refers to the state of the system when $\text{ShiftClock} = t$.

The bound on packet delay is based on the following theorem.

Theorem 1: In a time-shift scheduler for every active flow f

$$\text{ShiftClock} \leq T.f + L_{\max}/C - L.f/C.$$

Proof: Consider an active flow f and let p be the packet currently at the head of the queue of flow f. Let t be the time when p became the head of the queue of flow f and let s be the latest time, no later than t ($s \leq t$), such that one of the following action executions occurred:

1) a time shift occurred, i.e., ShiftClock was increased, and $s = \text{ShiftClock}$ after the increase;
2) a packet q from a flow g, $f \neq g$ was forwarded, where $T.q > T.p$;
3) a packet was received when all of the queues were empty.

In all three cases, at time s, for all active flows g

$$s \leq I.g \vee T.p < T.g. \tag{3}$$

In case 1 this holds because after the time shift $s = I_{\min} \leq I.g$. In case 2 since the packet chosen to be forwarded had $T.q > T.p$, then all active flows g have $T.g \geq T.q > T.p$. In case 3 the only active flow is the one whose packet is received at time s, and thus its ideal arrival time must be at least s.

Assume $T.p < s$. From (3) and $I.g < T.g$, for all active flows g at time s

$$T.p < T.g.$$

Furthermore, when any flow g becomes active after s

$$T.p < s < T.g.$$

Hence, $T.p < s$ is impossible because flow f is active at time t, $s \leq t$, with $T.f = T.p$. We must have instead that $s \leq T.p$.

Let us first assume that no time shift occurs after s and before p is forwarded.

From the definition of s, only packets from flows g with $T.g \leq T.p$ are forwarded after s and until p is forwarded. Thus, from Lemma 1 below (replacing u by $T.p$) and from (3), these packets, which include p, can be at most $(T.p - s) \cdot C$ bits. Since no time shift occurs after s, the last bit of packet p exits the output channel no later than time

$$s + (T.p - s) + L_{\max}/C$$

that is, no later than time $T.p + L_{\max}/C$. The term $L_{\max}/C$ is needed because for cases 1 and 2, we did not count the packet currently being in the output channel at time s.

If a time shift occurs after s and before p is forwarded, let s' be the time of the last time shift before p is forwarded. From the definition of a time shift, $s' \leq I.g$ for all active flows g at time s'. Also, since $s < s'$, only packets whose timestamps are at most $T.p$ are forwarded to the output channel until p is forwarded. Thus, from Lemma 1 (replacing u by $T.p$), these packets, which include p, can be at most $(T.p - s') \cdot C$ bits. Since no time shift occurs after s', the packet p exits the output channel no later than time

$$s' + (T.p - s') + L_{\max}/C.$$

that is, no later than time $T.p + L_{\max}/C$. Again, the term $L_{\max}/C$ is needed because we did not count the packet currently in the channel at time s'.

Recall that p is currently the head of the queue of flow f. Thus, $T.p = T.f$ until p is forwarded. Packet p will be forwarded to the output channel no later than time $T.f + (L_{\max} - L.f)/C$, and at this time either f becomes inactive or $T.f$ increases, which implies the theorem. □

Lemma 1: If $s \leq u$ and at time s, for all active flows f

$$s \leq I.f \ \vee \ u < T.f$$

then, starting from time s, the size of the packets forwarded to the output channel with a timestamp at most u have a total of at most $(u - s) \cdot C$ bits. □

The proof of Lemma 1 is found in the appendix.

Theorem 1 implies that each packet p will exit the output channel before ShiftClock reaches the value of its timestamp $T.p$ plus $L_{\max}/C$. It has been shown that the exit time of a packet in virtual clock scheduling is at most the packet's timestamp plus $L_{\max}/C$ [3], [24]. This is not directly comparable with the above bound for time-shift scheduling because the exit time in the time-shift scheduler is measured with respect to ShiftClock and not with respect to the real-time clock.

To show that the delay bound of the time-shift scheduler is no greater than the delay bound of virtual clock, we consider an alternative protocol as follows. Instead of shifting the clock forward δ s during a time shift, the timestamps of all flows (active or inactive) are reduced by δ s. Because the relative values of the flow timestamps with respect to each other and with respect to the clock is the same as in the time-shift scheduler, the order in which packets are forwarded, and hence the delay, remains the same. Furthermore, it is easy to show that Theorem 1 holds for this alternative protocol. Finally, note that in this alternative protocol the timestamp of each packet is at most the timestamp of the same packet in a virtual clock scheduler because the flow timestamps are being reduced. Thus, the delay bound for a time-shift scheduler is at most the delay bound for a virtual clock scheduler.

The delay bound of weighted fair queueing is similar to that of virtual clock [20]. Hence, the delay bound of time-shift scheduling is also similar to that of weighted fair queueing.

VI. Fairness

We next examine the fairness of a time-shift scheduler. To ensure fairness, the protocol given above must be changed slightly. To keep the difference between the timestamps of any pair of flows as small as possible, the scheduler should perform a time shift often, in particular, before a flow becomes active. This can be accomplished by having the scheduler execute its third action (if enabled) immediately before receiving a packet from an inactive flow.

We refer to the above scheduler as a fair time-shift scheduler.

Definition 1: An active flow f is a *minimum serviced flow* (MSF) if $I.f = I_{\min}$.

For a fair time-shift scheduler, since a time shift is executed always before receiving a packet from an inactive flow, it is easy to show that $I_{\min}$ increases monotonically. Thus, if flow f is an MSF, it remains an MSF until the next packet from f is forwarded.

The fairness of a fair time-shift scheduler is based on the following theorem.

Theorem 2: In a fair time-shift scheduler, for any pair f and g of active flows, there exists an MSF m such that

$$|T.f - T.g| \leq L.m/R.m + |L.f/R.f - L.g/R.g| + L_{\max}/C.$$

Proof: We prove the theorem by showing that the fair time-shift scheduler preserves the following invariant. For every flow f, one of the following conditions holds:

1) f is active, and $I.m \leq I.f \leq T.m + L_{\max}/C$ for some MSF m;
2) f is inactive, and

$$T.f \leq \text{ShiftClock} \ \vee \ T.f \leq I_{\min}$$

3) f is inactive, and $I.m \leq T.f \leq T.m$ for some MSF m.

The proof that the scheduler satisfies this invariant is found in the appendix. We next show that the invariant implies the theorem. For the theorem, we only need case 1 of the invariant. However, cases 2 and 3 are needed to show that case 1 holds when an inactive flow becomes active.

Consider two active flows f and g. From case 1 above, there exist two MSF flows m and m' such that

$$I.m \leq I.f \leq T.m + L_{\max}/C$$
$$I.m' \leq I.g \leq T.m' + L_{\max}/C.$$

Without loss of generality, assume that $T.m' \leq T.m$. Since m and m' are MSF, $I.m = I.m'$. Thus, from the definition of ideal arrival time, we obtain the following upper bounds for $T.f$ and $T.g$:

$$I.m + L.f/R.f \leq T.f \leq I.m + L.m/R.m + L.f/R.f + L_{\max}/C$$
$$I.m + L.g/R.g \leq T.g \leq I.m + L.m/R.m + L.g/R.g + L_{\max}/C.$$

The upper and lower bounds on $T.f$ and $T.g$ imply the theorem. □

The bound on the relative value of two flow timestamps given by Theorem 2 prevents flows that become active from "hogging" the output channel and denying service to flows that have exceeded their reserved rates. The virtual clock protocol has no similar bound.

The above bound is close to, but not as tight as, the bound provided by the self-clocking fair queueing protocol. However, the fairness bound above is achieved in conjunction with a delay bound that is significantly better than the delay bound of self-clocking fair queueing.

The fairness of the scheduler implies that when a packet p arrives at the head of the queue of its flow, it will exit the output channel within a time bound that is independent of the value of its timestamp or the timestamps of other flows, as shown next.

Theorem 3: In a fair time-shift scheduler, the packet at the head of the queue of flow f will exit the output channel in at most

$$L.f/R.f + L.m/R.m + L_{\max}/C \text{ s}$$

for some MSF m.

Proof: Let p be the packet at the head of the queue of f, and it became the head of the queue at time t. Let $T.f'$ be the timestamp of flow f before p becomes the head of the queue. We have two choices regarding how p became the head of the queue.

First, assume p was received when the queue of f was empty. If $I.p$ is assigned ShiftClock, then from Theorem 1, p will exit the output channel in $L.p/R.f + L_{\max}/C$ s. If $I.p$ is assigned $T.f'$, then from part 3 in the proof of Theorem 2 (parts 1 and 2 do not apply), we have

$$I.m \leq T.f' = I.p \leq T.m \tag{4}$$

for some MSF m. Since $I_{\min}$ is nondecreasing, only packets whose timestamps are in the range $[I.m, T.p]$ are forwarded after t, which, from the proof of Lemma 1, are at most $(T.p - I.m) \cdot C$ bits. Furthermore

$$\begin{aligned}(T.p - I.m) \cdot C &= (I.p + L.p/R.f - I.m) \cdot C \\ &\leq (T.m + L.p/R.f - I.m) \cdot C \\ &= (L.m/R.m + L.p/R.f) \cdot C.\end{aligned}$$

Thus, p will exit the output channel in $L.m/R.m + L.p/R.f + L_{\max}/C$ s, where $L_{\max}/C$ s comes from the time required for the packet forwarded before p to exit the output channel.

Assume now that p became the head of its queue when another packet from f was forwarded. Thus, $T.f' = I.p$. Let m be an MSF when p becomes the head of its queue. Thus, $I.m \leq I.p$. If $m = f$, then (4) holds. If $m \neq f$, then m was not chosen to be forwarded and, hence, $I.p = T.f' \leq T.m$, and (4) holds.

Thus, the same relation of the previous case holds, and p will exit the output channel within $L.m/R.m + L.p/R.f + L_{\max}/C$ s. □

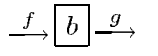
VII. End-to-End Delay Bounds

In this section we present the end-to-end delay bound for a flow traversing multiple time-shift schedulers. To do so, we borrow some results from flow theory [1], [2], which we overview next. The theorems are presented without proofs. The proofs may be found in [5].

A *flow* f is an infinite sequence $f.0, f.1, f.2, \cdots$, of nonnegative real numbers.

Informally, we divide time into very small and fixed-sized intervals, which we call instants. Each $f.i$ represents the number of bits that travel in flow f at instant i. We denote the sequence $f.0, f.1, \cdots, f.i$ by $f.(0, i)$.

As the flow traverses the network, it experiences queueing delays, which we represent with flow operators. A *flow operator* has an input flow f and an output flow g. At the ith instant, the operator inputs $f.i$ on its input flow, outputs $g.i$ on its output flow, and stores the remainder in an internal buffer. The content of the internal buffer at the ith instant is denoted $b.i$. The infinite sequence $b.0, b.1, b.2, \cdots$, is called the *buffer flow* of the flow operator.



Formally, a flow operator with an input flow f, an output flow g, and a buffer flow b is defined, for every $i = 0, 1, 2, \cdots$, as follows:

$$g.i \text{ is a value in the interval } F[f.(0, i), b.(i-1), i] \quad (5)$$

$$b.i = b.(i-1) + f.i - g.i \quad (6)$$

where $b.(-1) = 0$, and F is a function, called the *operation* of the flow operator, that returns an interval of real numbers.

Flow operators cannot output more than which they have received, and thus the operation of a flow operator is required to ensure the following flow *conservation property*:

(For every input flow f, output flow g, and buffer flow b, satisfying (5) and (6),
(For every i, $g.i \leq f.i + b.(i-1)$)
).

The buffer capacity B of a flow operator with input flow f is the smallest nonnegative real number that satisfies the following condition:

(For every output flow g and buffer flow b, satisfying (5) and (6),
(For every i, $b.i \leq B$)
).

The delay D of a flow operator with input flow f is the smallest nonnegative integer that satisfies the following condition:

(For every output flow g and buffer flow b, satisfying (5) and (6),
(For every i, $b_i \leq g.(i+1) + \cdots + g.(i+D)$)
).

To represent the changes that occur to a flow as it traverses its path in the computer network, we introduce linear networks of flow operators.

A finite sequence of flow operators $\langle \mathrm{Op}_0, \mathrm{Op}_1, \cdots, \mathrm{Op}_{n-1} \rangle$ is a *linear network* iff for each i, $0 \leq i < n-1$, the output flow of operator Op_i is the input flow of operator Op_{i+1}. The input flow of operator Op_0 is the input flow of the network and the output flow of operator Op_{n-1} is the output flow of the network.

The buffer flow c of a linear network $\langle \mathrm{Op}_0, \mathrm{Op}_1, \cdots, \mathrm{Op}_{n-1} \rangle$ is an infinite sequence $c.(-1), c.0, c.1, \cdots$, such that $c.(-1) = 0$ and for every $i = 0, 1, 2, \cdots$,

$$c.i = c.(i-1) + f.i - g.i$$

where f is the input flow of the linear network and g is the output flow of the linear network.

This definition of the buffer flow of a network coincides with the definition of the buffer flow of a flow operator where the input flow is f and the output flow is g. Hence, we define the buffer capacity and delay of a linear network in the same manner as was done previously for flow operators. It can be shown that the buffer capacity of a linear network is at most the sum of the buffer capacities of each of the flow operators in the linear network. A similar relation holds for the delay of the linear network [5].

The first flow operator we introduce is the R-limiter, where R is a positive real number. The operation of an R-limiter is defined as follows:

$$g_i = \begin{cases} R, & \text{if } b_{i-1} + f_i > R \\ b_{i-1} + f_i, & \text{if } b_{i-1} + f_i \leq R \end{cases}$$

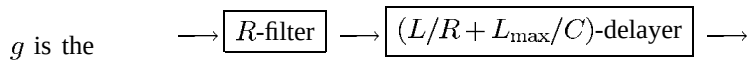
where f is the input flow, g is the output flow, and b is the buffer flow of the R-limiter. The R-limiter ensures that each element of its output flow is at most R. Basically, the R-limiter may be viewed as a constant-rate server that forwards its input flow to its output flow at exactly the rate R.

In virtual clock scheduling, the timestamp of a packet of a flow with reserved rate R is the time at which the packet would exit a constant-rate server that serves the flow at a rate R. A packet in virtual clock exits the output channel no later than the time indicated in its timestamp plus $L_{\max}/C$; that is, no later than the time at which it would exit an R-limiter plus $L_{\max}/C$.

Since the delay in a time-shift scheduler is at most that of virtual clock, a packet in a time-shift scheduler also exits no later than the time at which it would exit an R-limiter plus $L_{\max}/C$. Note that a packet may exit at a time much earlier than that at which it would exit an R-limiter.

This behavior can be represented by a flow operator called an R-filter, where R is a positive real number. An R-filter is a flow operator that outputs an arbitrary value at each instant, with the following restriction. If an R-limiter and an R-filter have the same input flow, then the sum of any prefix of the output flow of the R-filter is at least the sum of the same length prefix of the output flow of the R-limiter. That is, the R-filter may forward data faster than the R-limiter, and it is never allowed to lag behind the R-limiter. The formal definition of an R-filter may be found in [1] and [5].

From the definition of an R-filter, each packet in its input flow exits no later than the time it would exit an R-limiter. Thus, an R-filter can be used to represent the behavior of a single time-shift scheduler. However, in a time-shift scheduler a packet may exit up to $L_{\max}/C$ s after it would exit an R-limiter. Furthermore, in an R-filter, the first bit of a packet of size L may exit up to L/R instants earlier than the last bit of the packet. These bits should exit together, since packets are indivisible units of data and must be transmitted as a whole. Therefore, we represent a single time-shift scheduler with the following linear network, where L is the maximum packet size of the flow and R is the reserved rate of the flow.



A d-delayer, where d is positive integer, is a flow operator that delays its input flow in an arbitrary manner by at most d instants. Its formal definition may be found in [1] and [5]. The R-filter and d-delayer operators have the following useful properties.

Theorem 4:

1) The linear network $\langle R\text{-filter},\ R\text{-filter}\rangle$ is equivalent to a single R-filter.
2) The linear network $\langle d\text{-delayer},\ d'\text{-delayer}\rangle$ is equivalent to a single $(d+d')$-delayer. □

Note that, by induction on part 1, any linear network of R-filters is identical to a single R-filter.

Theorem 5:

1) When their input flow is the same, any output flow of the linear network $\langle d\text{-delayer},\ R\text{-filter}\rangle$ is also an output flow of the linear network $\langle R\text{-filter},\ d\text{-delayer}\rangle$.
2) The buffer capacity of $\langle R\text{-filter},\ d\text{-delayer}\rangle$ is at most the buffer capacity of a single R-filter plus $d \cdot R$. □

Assume that a flow with reserved rate R and maximum packet size L traverses a path of n time-shift schedulers. Let $L.i_{\max}$ and $C.i$ be, respectively, the upper bound on packet size and the capacity of the output channel of the ith scheduler along the path. The ith scheduler in this path is represented by the pair of flow operators $\langle R\text{-filter},\ (L/R + L.i_{\max}/C.i)\text{-delayer}\rangle$. The whole path consists of a linear network with n of these pairs.

Using Theorem 4 and part 1 of Theorem 5, it is easy to show that any output flow of the above linear network is also an output flow of the linear network $\langle R\text{-filter},\ D\text{-delayer}\rangle$, where

$$D = \sum_{i=0}^{n-1} (L/R + L.i_{\max}/C.i).$$

Thus, the delay of a path of time-shift schedulers is at most the delay of an $\langle R\text{-filter},\ D\text{-delayer}\rangle$ network. Since the delay of an R-filter is at most the delay of an R-limiter [5], the delay of the $\langle R\text{-filter},\ D\text{-delayer}\rangle$ network is at most the delay of an R-limiter plus D.

The delay of a flow through an R-limiter depends on the "burstiness" of the flow. That is, how much does the flow temporarily exceed its reserved rate R? There are several ways to characterize the burstiness of a flow [2], [6], [13]. One way to characterize burstiness is with the (m, R)-uniform property [2], which is defined as follows.

Let m be a positive integer and R be a positive real number. A flow f is (m, R)-uniform iff, for every $j = 0, 1, 2, \cdots$,

$$\sum_{i=j}^{j+m-1} f.i \le m \cdot R.$$

In this definition R can be regarded as an upper bound on the rate of the flow and m can be regarded as the interval over which the rate is averaged. The burstiness of the flow is measured by m: the larger m becomes, the burstier the flow may become.

The delay of an (m, R)-uniform flow through an R-limiter is at most m [5]. Thus, the total end-to-end delay along the path of n time-shift schedulers is at most $m + D$.

From Theorem 4 and part 1 of Theorem 5, the linear network consisting of the first j $\langle R\text{-filter},\ (L/R + L.i_{\max}/C.i)\text{-delayer}\rangle$ pairs, i.e., the first j time-shift schedulers, has a buffer capacity of at most the buffer capacity of the pair $\langle R\text{-filter},\ D'\text{-delayer}\rangle$, where

$$D' = \sum_{i=0}^{j-1} (L/R + L.i_{\max}/C.i).$$

Since the buffer capacity of the jth time-shift scheduler is at most the buffer capacity of the first j time-shift schedulers, then the buffer capacity of the jth scheduler is at most the buffer capacity of the pair $\langle R\text{-filter},\ D'\text{-delayer}\rangle$.

The buffer capacity of an R-filter is at most the buffer capacity of an R-limiter [5] and thus, from part 2 of Theorem 5, the buffer capacity of the jth time-shift scheduler is at most the buffer capacity of an R-limiter plus $D' \cdot R$. For an (m, R)-uniform flow, the buffer capacity of an R-limiter is $m \cdot R$ and thus, the buffer capacity of the jth time-shift scheduler is at most $m \cdot R + D' \cdot R$.

The upper bounds on the end-to-end delay and buffer capacities of a series of time-shift schedulers derived in this section are the same upper bounds on the end-to-end delay and buffer capacities reported for a series of virtual clock or weighted fair queueing schedulers [8], [12], [20]. Thus, the time-shift scheduler achieves the same end-to-end delay and buffer capacities, while at the same time being fair and efficient.

VIII. Related and Future Work

In time-shift scheduling, the end-to-end delay increases by $L/R + L_{\max}/C$ with each hop in the path to the destination. It is possible to decrease this delay for some flows at the expense of either increasing the delay of other flows or leaving some bandwidth unreserved. This has already been done in other protocols that do not ensure fairness [7], [9], [10]. In a future paper we will describe a schedulability test, similar to the one presented in [28], that allows time-shift scheduling to provide a delay bound independent of the reserved rate of the flow, while at the same time ensuring the fairness property presented in this paper.

Note that in the proof of the upper bound on packet delay (Theorem 1), we made no assumption about how often the scheduler performs a time shift. Furthermore, the theorem's proof can be relaxed to show that during a time shift, if the scheduler assigns a value smaller than $I_{\min}$ to the clock, then the theorem remains valid. Thus, the scheduler has a lot of freedom in manipulating the clock's value.

In a future paper we will describe a whole family of protocols that assign timestamps to packets and shift the clock periodically. The distinguishing factor of each member of the family is the extent to which the clock is shifted. All members of the protocol family will satisfy the rate-proportional delay property. We will argue that, contrary to popular belief, the clock does not always need to increase at least as fast as real time in order to maintain rate-proportional delay.

Appendix

Lemma 1: If $s \le t$, and at time s for all active flows f

$$s \le I.f \ \vee \ t < T.f$$

then, starting from time s, the size of the packets forwarded to the output channel with a timestamp at most t have a total

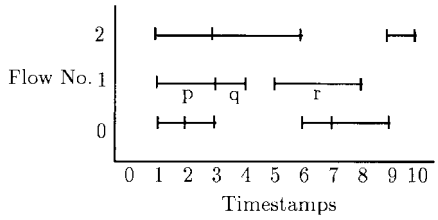


Fig. 4. Timestamps contained by each packet.

of at most $(t - s) \cdot C$ bits.

Proof: We associate a timestamp with each bit b of a packet p from flow f. We allow b to be a real number in the interval $[0, L.p]$. For bit b in packet p, its timestamp equals $I.p + b/R.f$. Thus, the first bit has a timestamp equal to $I.p$ and the last bit has a timestamp equal to the packet timestamp $T.p$. A timestamp u is contained by a packet p if a bit in the packet has timestamp u, i.e., $I.p \leq u \leq T.p$. A flow f contains timestamp u if some packet of f contains u.

For example, consider Fig. 4. The ideal arrival time and timestamp of each packet are denoted by a small vertical line. In flow 1, for its first packet p, we have $I.p = 1$ and $T.p = 3$. Similarly, we have $I.q = 3$, $T.q = 4$, $I.r = 5$, and $T.r = 8$. Thus, all timestamp values in the intervals [1, 4] and [5, 8] are contained by flow 1.

We begin by showing that if timestamp u is contained by a packet p of some flow f, then f is live at time u and, furthermore, the value of $R.f$ used to compute $T.p$ is the same value that $R.f$ has at time u. We have the following two cases.

- If p arrived at a time at most u, we have two cases. If p is in the queue at time u, then f is live at u. Since $R.f$ cannot change while f is live, the value of $R.f$ at time u is used to timestamp p. If p was forwarded before time u, then, since $T.p \geq u$, f is live at time u and $R.f$ cannot change until $\mathrm{ShiftClock} > u$.
- If p arrives after time u, then $\mathrm{queue}.f$ cannot become empty at any time starting at time u and until p arrives, since this would imply $I.p > u$, contradicting our choice of p. Hence, f is live at time u and $R.f$ cannot change until p is received and forwarded.

From the above observation, for packets p and q in Fig. 4, we have $I.p < T.p = I.q < T.q$, then both packets were assigned a timestamp using the same value for $R.1$. This is because flow 1 never ceases to be live from time $I.p$ up to $T.q$. Thus, for any interval $[u, v]$ contained by a flow f, the number of bits whose timestamps are in this interval is $R.f \cdot (v - u)$, where $R.f$ is the rate used to timestamp the packets containing the interval.

We next examine how many bits have a timestamp in the interval $[s, t]$. Starting from time s, find the latest time $s.0$, $s \leq s.0 \leq t$ such that no timestamp between s and $s.0$ is contained by a packet. In the figure, $s.0 = 1$ if we assume $s = 0$ and $t = 10$.

Next, find the largest timestamp $s.1$, $s.0 < s.1 \leq t$ such that if a flow does not contain a timestamp x in the interval $[s.0, s.1]$, then the flow does not contain any timestamps in the interval $[x, s.1]$. In Fig. 4 $s.1$ equals 5 because flow 1 does not contain any value in the interval [4, 5].

From the observations above, for any flow f that contains timestamps in $[s.0, s.1]$, at most $(s.1 - s.0) \cdot R.f$ bits contain timestamps in the interval $[s.0, s.1]$, where $R.f$ is the rate of flow f at time $s.0$ and, furthermore, f is live at time $s.0$. From (2), the sum of the rates of the live flows at time $s.0$ is at most C and, hence, at most $(s.1 - s.0) \cdot C$ bits contain timestamps in the interval $[s.0, s.1]$.

We next find a timestamp $s.2$, whose relationship to $s.1$ is similar to the relationship between $s.1$ and $s.0$. The same argument shows that at most $(s.2 - s.1) \cdot C$ bits contain timestamps in the interval $[s.1, s.2]$. By repeating the above steps until we reach a timestamp $s.n$ where $s.n = t$, we have that the total number of bits whose timestamp is in the interval $[s, t]$ is at most $(t - s) \cdot C$.

Since we are given that for all active flows f at time s, $s \leq I.f \ \vee \ t < T.f$, then each packet p forwarded starting from time s with timestamp at most t has $I.p \geq s$ and, hence, the lemma holds. □

Lemma 2: The following is an invariant of the fair time-shift scheduler. For every flow f, one of the following holds:

1) f is active, and $I.m \leq I.f \leq T.m + L_{\max}/C$ for some MSF m;
2) f is inactive, and

$$T.f \leq \mathrm{ShiftClock} \ \vee \ T.f \leq I_{\min}$$

3) f is inactive, and $I.m \leq T.f \leq T.m$ for some MSF m.

Proof: We show that if the protocol is in a state satisfying any of the three cases above, then it will continue to satisfy the same case or satisfy one of the remaining two cases.

We begin with case 1.

Notice that once a flow m has $I.m = I_{\min}$, this continues to hold until flow m forwards a packet. That is, when a flow k becomes active, a time shift is performed and $I_{\min} \leq \mathrm{ShiftClock} \leq I.k$. Hence, $I.m = I_{\min} \leq I.k$. Because $I_{\min}$ does not decrease, case 1 continues to hold until a packet from either m or f is forwarded.

Assume that a packet from flow f is forwarded, and flow f is the only active flow. Note that $m = f$ in this case. If $\mathrm{queue}.f$ becomes empty, ShiftClock is increased, if necessary, so that case 2 holds. If $\mathrm{queue}.f$ does not become empty, then case 1 continues to hold with $m = f$.

Assume that f is not the only active flow, and a packet from f is forwarded. Consider first that $f \neq m$ in case 1. Because the packet from m was not chosen to be forwarded, $T.f \leq T.m$ before updating $T.f$. Also, we are given that $I.m \leq I.f$, and thus $I.m \leq I.f < T.f \leq T.m$. After $T.f$ is updated from the next packet in the queue of f ($I.f$ is now the previous value of $T.f$), we have $I.m \leq I.f \leq T.m$. Thus, case 1 continues to hold. If no more packets from f remain, then $T.f$ is not updated and thus case 3 holds.

Assume next that $f = m$ in case 1. Let k be an MSF after the packet from f is forwarded and $T.f$ is updated. If f is an MSF in this case, then case 1 continues to hold with $f = m$. Otherwise, $k \neq f$, and because k was not chosen to be forwarded, $T.f \leq T.k$ before forwarding the packet from f, and thus $T.k \geq I.f$ after $T.f$ is updated. Furthermore, because k is an MSF, $I.k \leq I.f$. Thus, case 1 continues to hold, with k replacing m.

The above assumes that packets from f remain and $T.f$ is updated. If this is not the case, then, because k was not chosen to be forwarded, $T.f \leq T.k$. If $I.k \leq T.f$, then case 3 holds with k replacing m. Otherwise, $T.f < I.k = I_{\min}$, and case 2 holds.

Assume now that a packet from m is forwarded, and $m \neq f$. Let k be an MSF after the packet is forwarded. Let $T.m'$ be the timestamp of m just before the packet is forwarded. Because k is an MSF, then $I.k \leq I.f$ after the packet is forwarded. If $k \neq m$, then, because k was not chosen to be forwarded, $T.m' \leq T.k$. If $k = m$, then, after the packet is forwarded, $T.m' = I.k < T.k$. Thus, from case 1, from $T.m' \leq T.k$, and from k being an MSF, we have $I.k \leq I.f \leq T.k + L_{\max}/C$ after the packet is forwarded. Hence, case 1 holds with k replacing m.

Consider now case 2.

Recall that $I_{\min}$ and $\mathrm{ShiftClock}$ increase monotonically. Hence, case 2 can only be falsified if f becomes active, which we consider next.

If no flow is active and f becomes active, then case 1 holds with $m = f$. If other flows were active when the packet from f is received, then a time shift is performed immediately before receiving the packet from f. Let m be an MSF after the time shift. From Theorem 1 and from the time shift

$$I.m \leq \mathrm{ShiftClock} \leq T.m + L_{\max}/C - L.m/C. \quad (7)$$

Let $T.f'$ be the value of the timestamp of f before the packet is received. Recall that $I.f$ is assigned $\max(\mathrm{ShiftClock}, T.f')$. If $I.f$ is assigned ShiftClock, then case 1 holds by (7). If $I.f$ is assigned $T.f'$ (i.e., $\mathrm{ShiftClock} < T.f'$), then from case 2 we have $\mathrm{ShiftClock} < T.f' \leq I_{\min} = I.m$, which contradicts (7) and, hence, will not occur.

Consider now case 3.

This case is affected if flow f becomes active or if a packet from flow m is forwarded. If flow f becomes active, then a time shift is performed immediately before receiving the packet from f, and (7) holds. If $I.f$ is assigned ShiftClock, then case 1 holds from (7). If $I.f$ is assigned the previous value of $T.f$, then from case 3

$$I.m \leq I.f \leq T.m$$

which implies that case 1 holds.

If a packet from flow m is forwarded, then let k be an MSF after the packet is forwarded and let $T.m'$ be the timestamp of m before the packet is forwarded. If $k \neq m$, then, since k was not chosen to be forwarded, $T.m' \leq T.k$. If $k = m$, then, after the packet is forwarded, $T.m' = I.k < T.k$. Thus, from case 3, $T.f \leq T.k$. If after the packet of m is forwarded, $I.k < T.f$, then case 3 holds. Otherwise, $T.f \leq I.k = I_{\min}$, and case 2 will hold. If no active flow remains after forwarding the packet from m, then ShiftClock is advanced such that $\mathrm{ShiftClock} \geq T.m' \geq T.f$ and, hence, case 2 holds.

All cases of the invariant are either preserved or another case holds. Thus, the invariant is true. □

ACKNOWLEDGMENT

The authors would like to thank the referees for their constructive criticism, which improved the quality of this paper.

REFERENCES

[1] J. Cobb and M. Gouda, "Flow theory," *IEEE/ACM Trans. Networking*, vol. 5, pp. 661–674, Oct. 1997.

[2] ——, "Flow theory: Verification of rate-reservation protocols," in *IEEE Int. Conf. Network Protocols*, San Francisco, CA, 1993, pp. 198–205.

[3] J. Cobb, M. Gouda, and A. El-Nahas, "Flow timestamps," in *Annu. Joint Conf. Information Sciences*, 1995.

[4] ——, "Time-shift scheduling: Fair scheduling of flows in high-speed networks," in *IEEE Int. Conf. Network Protocols*, Columbus, OH, 1996, pp. 6–13.

[5] J. Cobb, "Flow theory and the analysis of timed-flow protocols," Ph.D. dissertation, Univ. Texas, Austin, TX, May 1996.

[6] R. L. Cruz, "A calculus for network delay, Part I: Network elements in isolation," *IEEE Trans. Inform. Theory*, vol. 37, pp. 114–131, Jan. 1991.

[7] N. R. Figueira and J. Pasquale, "Leave-in-time: A new service discipline for real-time communications in a packet-switching data network," in *Proc. ACM SIGCOMM*, Cambridge, MA, 1995, p. 207.

[8] ——, "An upper bound on delay for the virtual clock service discipline," *IEEE/ACM Trans. Networking*, vol. 3, pp. 399–408, Aug. 1995.

[9] D. Ferrari and D. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 368–379, Apr. 1990.

[10] D. Ferrari and H. Zhang, "Rate controlled static priority queueing," in *Proc. IEEE INFOCOM*, San Francisco, CA, 1993, pp. 227–236.

[11] D. Gall, "A video compression standard for multimedia applications," *Commun. ACM*, vol. 34, no. 4, pp. 47–58, Apr. 1991.

[12] P. Goyal, S. Lam, and H. Vin, "Determining end-to-end delay bounds in heterogeneous networks," in *NOSSDAV Workshop*, Durham, NH, 1995, pp. 287–298.

[13] S. J. Golestani, "A stop-and-go queueing framework for congestion management," in *Proc. ACM SIGCOMM*, Philadelphia, PA, 1990, pp. 8–18.

[14] ——, "A self-clocking fair-queueing scheme for broadband applications," *Proc. IEEE INFOCOM*, Toronto, Ont., Canada, 1994, pp. 636–646.

[15] M. Gouda, "Protocol verification made simple," *Comput. Networks ISDN Syst.*, vol. 25, pp. 969–980, 1993.

[16] ——, *The Elements of Network Protocols*, to be published.

[17] S. Keshav, "A control theoretic approach to flow control," in *Proc. ACM SIGCOMM*, Zurich, Switzerland, 1991, pp. 3–15.

[18] C. R. Kalmanek, H. Kanakia, and S. Keshav, "Rate controlled servers for very high-speed networks," in *Proc. IEEE GLOBECOM*, San Diego, CA, 1990, pp. 300.3.1–300.3.9.

[19] H. Kanakia and P. Mishra, "A hop-by-hop rate based congestion control scheme," in *Proc. ACM SIGCOMM*, Baltimore, MD, 1992, pp. 112–121.

[20] A. K. J. Parekh, "A generalized processor sharing approach to flow control in integrated services networks," Lab. Inform. Decision Syst., Massachusetts Inst. Technol., Cambridge, Tech. Rep. LIDS-TH-2089, Feb. 1992.

[21] A. K. J. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Trans. Networking*, vol. 1, pp. 344–357, June 1993.

[22] I. R. Philp and J. W. S. Liu, "End-to-end scheduling in real-time packet switched networks," in *IEEE Int. Conf. Network Protocols*, Columbus, OH, 1996, pp. 23–30.

[23] K. K. Ramakrishnan and J. Raj, "A binary feedback scheme for congestion avoidance in computer networks," *ACM Trans. Comput. Syst.*, vol. 8, no. 2, pp. 158–181, May 1990.

[24] G. Xie and S. Lam, "Delay guarantee of virtual clock server," *IEEE/ACM Trans. Networking*, vol. 3, pp. 683–689, Dec. 1995.

[25] L. Zhang, "Virtual clock: A new traffic control algorithm for packet-switched networks," *ACM Trans. Comput. Syst.*, vol. 9, no. 2, pp. 101–124, May 1991.

[26] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proc. ACM SIGCOMM*, Zurich, Switzerland, 1991, pp. 113–122.

[27] H. Zhang, "Service disciplines for guaranteed performance service in packet switching networks," *Proc. IEEE*, vol. 83, pp. 1374–1396, Oct. 1995.

[28] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet switched networks," *IEEE Trans. Commun.*, vol. 42, pp. 1096–1105, Mar. 1994.

Jorge A. Cobb received the B.S. degree (highest hons.) from the University of Texas, El Paso, in December 1987, and the M.A. and Ph.D. degrees from the University of Texas, Austin, in the summer of 1989 and the spring of 1990, respectively.

In the fall of 1995 he joined the faculty of the Department of Computer Science, University of Houston, Houston, TX, where he is currently an Assistant Professor. He was previously with AT&T Information Systems, Denver, CO. His main research interest is computer networking, with an emphasis on real-time scheduling and mobile computing. His research interests also include parallel and distributed computing.

Dr. Cobb received the AT&T Bell Laboratories Scholarship in 1992.

Mohamed G. Gouda received the B.A. degree in engineering and the B.A. degree in mathematics from Cairo University, Cairo, Egypt, the M.A. degree in mathematics from York University, Toronto, Ont., Canada, and the M.S. and Ph.D. degrees in computing science from the University of Waterloo, Waterloo, Ont., Canada.

He later moved to the U.S., where he worked for the Honeywell Corporate Technology Center for three years. In 1980 he joined the University of Texas, Austin, where he currently holds the Mike A. Myers Centennial Professorship in computing science. He spent one summer at Bell Laboratories, Murray Hill, NJ, one summer at MCC, Austin, TX, and one winter at the Eindhovern Technical University, The Netherlands. His primary research area is distributed and concurrent computing. In this area he has been working on abstraction, nondeterminism, atomicity, convergence, stability, formality, correctness, efficiency, scientific elegance, and technical beauty (not necessarily in that order). He was the founding Editor-in-Chief of the journal *Distributed Computing*, published by Springer-Verlag in 1985. He is the author of *Elements of Network Protocol Design* (New York: Wiley, to be published). This textbook is the first one where network protocols are specified using a formal abstract notation.

Dr. Gouda is an original member of the Austin Tuesday Afternoon Club. He received the 1993 Kuwait Award in Basic Sciences. He was the first Program Committee Chairman for the International Conference on Network Protocols, established by the IEEE in 1993, and the the first Program Committee Chairman for the Symposium on Advances in Computers and Communications, established by the IEEE and held in Egypt in 1995.

Amal El-Nahas received the B.Sc. and M.Sc. degrees in computer science from Alexandria University, Alexandria, Egypt, in 1986 and 1990, respectively. She pursued her Ph.D. studies under a joint supervision between Alexandria University and the University of Texas, Austin, and she received the Ph.D. degree in 1996 from Alexandria University in 1996.

Since 1996 she has been with the Department of Computer Science, Alexandria University, Alexandria, Egypt, as an Assistant Professor. Her main research interests include network protocols, quality-of-service guarantees for real-time applications, and wireless network protocols.